OIPE JC100
OCT 2 6 2001
PATENT & TRADEMARK OFFICE

Exhibit A

Communication Drivers

The Microsoft Windows communications driver provides a set of functions that Windows can use to open communication ports, set communication configurations, read and write characters, and retrieve error and status information. This topic describes the communication-driver functions.

The following topics provide useful information about the communication driver:

About the Communications Driver
Base Address and IRQ Selection
16550a UART FIFO Buffer
CommWriteString and EnableNotification Functions
Communication Escapes
Baud Rate Indexes
Communication Functions and Structures

About the Communications Driver

The communication driver is a dynamic-link library (DLL) containing functions that support opening, reading from, and writing to communications devices. Although the communication driver exports several functions, Windows-based applications do not directly call these functions. Instead, the applications call functions in the USER module, such as the **OpenComm** function, which in turn call the communications driver.

Unlike other Windows-based drivers, the communications driver does not include the **Enable** and **Disable** functions, although it does include the **WEP** function.

Base Address and IRQ Selection

The Windows 3.1 communications driver (COMM.DRV) accesses serial ports COM1, COM2, COM3, and COM4 using base-address values specified in the BIOS data area of a computer. If the BIOS data area does not specify values for physical ports, Windows 3.1 will use the base address, 0x03E8, and IRQ4 for COM3, but the base addresses and IRQs for the other ports must be set by the user.

If the BIOS data area does not specify a value for a physical port, the user can set the base address and IRQ values using Control Panel. Control Panel displays an Advanced settings dialog box for each port. The dialog box contains a selection box for base addresses and spin controls for IRQ settings. The selected values are recorded as **COM*x*Base** and **COM*x*IRQ** settings in the [386Enh] section of the SYSTEM.INI file.

The **COM*x*Base** and **COM*x*IRQ** settings are used for both standard- and 386 enhanced-mode operation. The **COM*x*Base** setting is used only if the BIOS data area does not specify a value for the port; this setting never overrides the BIOS data area values.

16550a UART FIFO Buffer

If a computer uses the 16550a Universal Asynchronous Receiver Transmitter (UART) for its communication ports, the communications driver will enable the onboard 16-byte first in, first out (FIFO) buffer allowing Windows to perform reliable serial communications at speeds of 9600 baud and higher. (Many computers that do not enable this buffer experience loss of characters at 9600 baud, and most cannot communicate at speeds higher than 9600 baud.)

Before enabling the FIFO buffer, the communications driver checks the SYSTEM.INI file to determine whether the user wants the buffers enabled. The COMxFIFO settings in the [386Enh] section of the SYSTEM.INI file specify whether the buffer for a given port should be enabled or disabled. If the value of the setting is 1, the driver enables the FIFO buffer; otherwise, it disables the buffer. If no setting is specified, the driver will enable the buffer by default.

The virtual-communications device (VCD) does not virtualize the 16550a for non-Windows applications.

CommWriteString and EnableNotification Functions

The communications driver exports the **CommWriteString** and **EnableNotification** functions. The **CommWriteString** function writes a string of one or more bytes to the given communications device.

The **EnableNotification** function enables or disables port status notifications. If notifications are enabled, the communications driver posts a **WM_COMMNOTIFY** message to a given window on certain events. This eliminates the need for communications applications to set timers and use the **GetCommError**, **GetCommEventMask**, and **SetCommEventMask** functions to monitor port status changes.

int CommWriteString(*cid, pbuf, size***)**

int *cid;*
LPSTR *pbuf;*
int *size;*

The **CommWriteString** function transmits a string of characters using the specified communications device. USER calls this function whenever an application calls the **WriteComm** function (USER.205).

Parameter	**Description**
cid	Identifies the communication device.
pbuf	Points to the buffer containing the bytes to write.
size	Specifies the number of bytes to write.

Returns

The return value is the number of bytes actually written.

Comments

The export ordinal for this function is 19.

During initialization, USER checks for this function to determine whether the driver supports the extended functions new to Windows 3.1. If the function is not present, Windows assumes that the driver is a Windows 3.0 driver and makes sure that all interaction with the driver is compatible with Windows 3.0.

int EnableNotification(*cid, hWnd, wInTrigger, wOutTrigger***)**

int *cid;*
HWND *hWnd;*
WORD *wInTrigger;*
WORD *wOutTrigger;*

The **EnableNotification** function enables or disables communications message posting. When enabled, the driver posts the **WM_COMMNOTIFY** message to the specified window. USER calls this function when an application calls the **EnableCommNotification** function (USER.245).

Parameter	Description
cid	Identifies the communication device.
hWnd	Identifies the window to receive the **WM_COMMNOTIFY** message. If this parameter is NULL, the function disables the notification.
wInTrigger	Specifies the minimum number of bytes to be received in the communication device's input buffer before receive notification is sent.
wOutTrigger	Specifies the maximum number of bytes to remain in the communication device's output buffer before transmit notification is sent.

Returns

The return value is TRUE if successful. Otherwise, the return value is FALSE.

Comments

The export ordinal for this function is 100.

The **WM_COMMNOTIFY** message has the following parameters.

Parameter	Description
wParam	Specifies the communication-device identifier (the *cid* parameter).
HIWORD(*lparam*)	Not used; must be zero.
LOWORD(*lparam*)	Specifies the notification status. It can be one of the following values.

Value	Meaning
CN_EVENT	An event enabled in the communication device's event mask (specified by the **SetCommEventMask** function) has occurred. The application should call the function **GetCommEventMask** to determine what event has occurred, and to clear the event. This status is sent when the communication device's event word changes. The application clears the appropriate event to ensure notification on subsequent events.
CN_RECEIVE	At least *wInTrigger* bytes are in the communication device's input buffer, or at least 1 byte is in the input buffer. Additionally, no more have been received before the end of an internal timeout period. The number of bytes in the input buffer must be lower than *wInTrigger* bytes before this message will be sent again.
CN_TRANSMIT	Fewer than *wOutTrigger* bytes remain in the communication device's output buffer to be transmitted. The number of bytes in the output buffer must exceed *wOutTrigger* bytes before this message will be sent again.

The communication device event may be a line-status or printer error. Applications can determine the cause by using the **GetCommError** function immediately after the **GetCommEventMask** function

(USER.209).

int CommWriteString(*cid, pbuf, size***)**

int *cid;*
LPSTR *pbuf;*
int *size;*

The **CommWriteString** function transmits a string of characters using the specified communications device. USER calls this function whenever an application calls the **WriteComm** function (USER.205).

Parameter	Description
cid	Identifies the communication device.
pbuf	Points to the buffer containing the bytes to write.
size	Specifies the number of bytes to write.

Returns

The return value is the number of bytes actually written.

Comments

The export ordinal for this function is 19.

During initialization, USER checks for this function to determine whether the driver supports the extended functions new to Windows 3.1. If the function is not present, Windows assumes that the driver is a Windows 3.0 driver and makes sure that all interaction with the driver is compatible with Windows 3.0.

Communication Escapes

The communications driver supports the RESETDEV, GETBASEIRQ, GETMAXLPT, and GETMAXCOM communication escapes in its **cextfcn** function. These communication escapes reset the printer (assert the reset line) and retrieve parallel- and serial-port identifiers.

Baud-Rate Indexes

The communications driver supports very high baud rates, such as 128,000 and 256,000, by interpreting the **BaudRate** member of the DCB structure as a baud-rate index whenever the high byte of the member is 0xFF. In such cases, **BaudRate** can be one of the following values.

BaudRate	Value
110	CBR_110 (0xFF10)
300	CBR_300 (0xFF11)
600	CBR_600 (0xFF12)
1200	CBR_1200 (0xFF13)
2400	CBR_2400 (0xFF14)
4800	CBR_4800 (0xFF15)
9600	CBR_9600 (0xFF16)
14,400	CBR_14400 (0xFF17)
19,200	CBR_19200 (0xFF18)
38,400	CBR_38400 (0xFF1B)
56,000	CBR_56000 (0xFF1F)
128,000	CBR_128000 (0xFF23)
256,000	CBR_256000 (0xFF27)

Note The CBR_ values are for standardization; drivers are not required to support all indexed baud rates.

If the high byte of the **BaudRate** member is not 0xFF, **BaudRate** specifies the actual baud rate for the communications device. In other words, values in the range 2 through 65,279 (0xFEFF) are interpreted as baud rate values not as indexes. This ensures compatibility with existing communications drivers.

```
typedef struct {
   char  Id;            /* internal device ID            */
   ushort Baudrate;      /* operating speed               */
   char  ByteSize;      /* transmit/receive byte size    */
   char  Parity;        /* 0,1,2,3, or 4                 */
   char  StopBits;      /* number of stop bits           */
   ushort RlsTimeout;    /* timeout for RLSD to be set    */
   ushort CtsTimeout;    /* timeout for CTS to be set     */
   ushort DsrTimeout;    /* timeout for DSR to be set     */
   ushort fBinary: 1;    /* binary-mode flag              */
   ushort fRtsDisable: 1; /* disable RTS                  */
   ushort fParity: 1;    /* enable parity checking        */
   ushort fDummy: 5;
   ushort fOutX: 1;      /* enable output XON/XOFF        */
   ushort fInX: 1;       /* enable input XON/XOFF         */
   ushort fPeChar: 1;    /* enable parity-error replacement */
   ushort fNull: 1;      /* enable null stripping         */
   ushort fChEvt: 1;     /* enable Rx character event     */
   ushort fDtrflow: 1;   /* enable DTR flow control       */
   ushort fRtsflow: 1;   /* enable RTS flow control       */
   ushort fDummy2: 1;
   char  XonChar;       /* transmit/receive XON character  */
   char  XoffChar;      /* transmit/receive XOFF character */
   ushort XonLim;        /* transmit XON threshold        */
   ushort XoffLim;       /* transmit XOFF threshold       */
   char  PeChar;        /* parity error replacement character */
   char  EofChar;       /* end-of-input character        */
   char  EvtChar;       /* event-generating character    */
   ushort TxDelay;       /* amount of time between characters */
} DCB;
```

The **DCB** structure contains the RS-232 configuration parameters for a communication device.

Member	Description
Id	Specifies the device ID byte (COM1 = 0, COM2 = 1, and so on). This is also the value returned by the **cOpen** function, when successful.
Baudrate	Specifies the operating speed; any baud rate supported by the hardware.
ByteSize	Specifies the transmitting and receiving byte size; normally in the range 4 through 8.
Parity	Specifies the parity setting. The value can be one of the following values.

Value	Meaning
0	None
1	Odd
2	Even
3	Mark
4	Space

StopBits Specifies the number of stop bits. The value can be one of the following values.

Value	Meaning
0	1 stop bit
1	1.5 stop bits
2	2 stop bits

RlsTimeout Specifies the amount of time, in milliseconds, to wait for receiving-line-signal detect

	(RLSD) to become high. RLSD flow control can be achieved by specifying infinite timeout (0xFFFF).
CtsTimeout	Specifies the amount of time, in milliseconds, to wait for clear-to-send signal (CTS) to become high. CTS flow control can be achieved by specifying infinite timeout (0xFFFF).
DsrTimeout	Specifies the amount of time, in milliseconds, to wait for data-set-ready (DSR) to become high. DSR flow control can be achieved by specifying infinite timeout (0xFFFF).
fBinary	Specifies the binary-mode flag (0 is ASCII mode, 1 is binary). In ASCII mode, the end-of-file character (EOFCHAR) is recognized and remembered as the end of received data.
fRtsDisable	Disables the receive-transmission signal (RTS) line for as long as this device is open, if set. Normally, RTS is enabled when the device is opened and disabled when closed.
fParity	Enables parity checking, if set.
fOutX	Indicates that XON/XOFF flow control is to be used during transmission, if set. The transmitter halts when it receives an XOFF character, and starts again when it receives an XON character.
fInX	Indicates that XON/XOFF flow control is to be used during reception, if set.
fPeChar	Indicates that characters received with parity errors are to be replaced with the specified parity-checking characters (PECHAR), if set.
fNull	Specifies that the received null characters are to be discarded, if set.
fChEvt	Indicates that the reception of event-checking characters (EVTCHAR) are to be flagged as an event, if set.
fDtrFlow	Indicates that the data-terminal-ready signal (DTR) is to be used for receive flow control, if set.
fRtsflow	Indicates that the receive-transmission signal (RTS) is to be used for receive flow control, if set.
XonChar	Specifies the XON character for both transmit and receive.
XoffChar	Specifies the XOFF character for both transmit and receive.
XonLim	Specifies the threshold value for receive queue. When the receive queue comes within 10 characters of being full, it transmits an XOFF character. When the queue comes within 10 characters of being empty, an XON character will be transmitted.
XoffLim	Specifies the threshold value for send queue. When the number of characters in the receive queue exceeds this value, an XOFF character is sent (if XOFF flow control is enabled) and the data-terminal-ready signal (DTR) is dropped (if enabled).
PeChar	Specifies the character to be used as replacement when a parity error occurs.
EofChar	Specifies the character that signals the end of the input.
EvtChar	Specifies the character that triggers an event flag.
TxDelay	Specifies the minimum amount of time that must pass between transmission of characters.

See Also

getdcb, inicom, setcom

Communication-Driver Functions

cclrbrk	Restore transmission
cevt	Sets the event mask
cevtGet	Retrieves the event mask
cextfcn	Carry out extended function
cflush	Flushes queues
CommWriteString	Transmits a string of bytes
csetbrk	Breaks transmission
ctx	Transmit immediately
EnableNotification	Enable/disables communication notification
getdcb	Retrieves device-control block
inicom	Initialize communications device
ReactivateOpenCommPorts	Reactivates communications ports
reccom	Read a byte
setcom	Set communications block
setque	Set transmit and recieve queues
sndcom	Transmit a byte
stacom	Retrieves error code and device status
SuspendOpenCommPorts	Suspends communications ports
trmcom	Closes the device
COMSTAT	Communications Status Structure
DCB	Device Control Block Structure
qdb	Queue Definition Block

int cclrbrk(*cid*)

int *cid;*

The **cclrbrk** function restores character transmission and places the communications device in a nonbreak state. USER calls this function whenever an application calls the **ClearCommBreak** function (USER.211).

Parameter	Description
cid	Identifies the communications device.

Returns

The return value is zero if the function is successful. Otherwise, the return value returns -1.

Comments

The export ordinal for this function is 14.

See Also

csetbrk

LPWORD cevt(*cid, evtmask***)**

int *cid;*
int *evtmask;*

The **cevt** function enables events in the event mask of the specified communications device. USER calls this function whenever an application calls the **SetCommEventMask** function (USER.208).

Parameter	Description
cid	Identifies the communications device.
evtmask	Specifies which events are to be enabled. This parameter can be any combination of the following values.

Value	Meaning
EV_BREAK	Enables detection of a break upon input.
EV_CTS	Enables detection of the clear-to-send (CTS) signal.
EV_DSR	Enables detection of the data-set-ready (DSR) signal.
EV_ERR	Enables detection of a line-status error. Line-status errors are CE_FRAME, CE_OVERRUN, and CE_RXPARITY.
EV_PERR	Enables detection of a printer error on a parallel device. Errors are CE_DNS, CE_IOE, CE_LOOP, and CE_PTO.
EV_RING	Indicates the state of ring indicator during the last modem interrupt. (Use EV_RINGTE to detect when a phone ring has occurred.)
EV_RLSD	Enables detection of the receive-line-signal-detect (RLSD) signal.
EV_RXCHAR	Enables detection of any character received and placed in the receive queue.
EV_RXFLAG	Enables detection of the event character received and placed in the receive queue. The event character is specified in the **EvtChar** member of the **DCB** structure.
EV_TXEMPTY	Enables detection of when the last character in the transmit queue is sent.

Returns

The return value is a pointer to a 16-bit buffer if the function is successful. Otherwise, the function returns zero if there is an error.

Comments

The export ordinal for this function is 11.

The communications driver sets one or more bits in the returned buffer whenever one of the events specified by the *evtmask* parameter occurs.

WORD cevtGet(*cid, evtmask***)**

int *cid;*
int *evtmask;*

The **cevtGet** function retrieves and then clears the event mask for a communications device. USER calls this function whenever an application calls the **GetCommEventMask** function (USER.209).

Parameter	Description
cid	Identifies the communications device.
evtmask	Specifies which events in the current event mask to disable. This parameter can be any combination of the following values.

Value	Meaning
EV_BREAK	Enables detection of a break upon input.
EV_CTS	Enables detection of the clear-to-send (CTS) signal.
EV_DSR	Enables detection of the data-set-ready (DSR) signal.
EV_ERR	Enables detection of a line-status error. Line-status errors are CE_FRAME, CE_OVERRUN, and CE_RXPARITY.
EV_PERR	Enables detection of a printer error on a parallel device. Errors are CE_DNS, CE_IOE, CE_LOOP, and CE_PTO.
EV_RING	Indicates the state of ring indicator during the last modem interrupt. (Use EV_RINGTE to detect when a phone ring has occurred.)
EV_RLSD	Enables detection of the receive-line-signal-detect (RLSD) signal.
EV_RXCHAR	Enables detection of any character received and placed in the receive queue.
EV_RXFLAG	Enables detection of the event character received and placed in the receive queue. The event character is specified in the **EvtChar** member of the <u>**DCB**</u> structure.
EV_TXEMPTY	Enables detection of when the last character in the transmit queue is sent.

Returns

The return value is the current event word as set by the **cevt** function.

Comments

The export ordinal for this function is 12.

See Also

cevt

LONG cextfcn(*cid*, *fcn***);**

int *cid;*
int *fcn;*

The **cextfcn** function carries out the extended communications function specified by the *fcn* parameter. USER calls this function when an application calls the **EscapeCommFunction** function (USER.214).

Parameter	Description
cid	Identifies the communications device.
fcn	Specifies the extended function to carry out. It can be one of the following values.

Value	Meaning
CLRDTR	Clears the data-terminal-ready (DTR) signal.
CLRRTS	Clears the request-to-send (RTS) signal.
GETBASEIRQ	Returns the base-port address and IRQ setting for the COM port specified by the *cid* parameter. The low 16 bits of the return address specifies the base-port address, and the high address specifies the IRQ selection. If the high 16 bits is -1, then the port does not exist; if it is zero, the installed COMM driver does not support this escape.
GETMAXBAUD	Returns a constant that indicates the highest baud rate supported by the port specified by the *cid* parameter. The following constants may be returned.

Value	Meaning
CBR_110	The highest baud rate is 110.
CBR_300	The highest baud rate is 300.
CBR_600	The highest baud rate is 600.
CBR_1200	The highest baud rate is 1200.
CBR_2400	The highest baud rate is 2400.
CBR_4800	The highest baud rate is 4800.
CBR_9600	The highest baud rate is 9600.
CBR_19200	The highest baud rate is 19,200.
CBR_38400	The highest baud rate is 38,400.
CBR_56000	The highest baud rate is 56,000.
CBR_128000	The highest baud rate is 128,000.
CBR_256000	The highest baud rate is 256,000.

GETMAXCOM	Returns the maximum COM port identifier supported by the communications driver. This value ranges from 0x00 to 0x7F, such that 0x00 corresponds to COM1, 0x01 to COM2, 0x02 to COM3, and so on. The communications driver included in Windows 3.1 supports COM1 through COM4, and will always return 0x03.
GETMAXLPT	Returns the maximum LPT port ID supported by the system. This value ranges from 0x80H to 0xFFH, such that 0x80H corresponds to LPT1, 0x81H to LPT2, 0x82H to LPT3, and so on.
RESETDEV	Resets the printer device (that is, assert the reset line) if the *cid* parameter specifies an LPT port. No function is performed if *cid* specifies a COM port.
SETDTR	Sets the data-terminal-ready (DTR) control line on.
SETRTS	Sets the request-to-send (RTS) control line on.

SETXOFF	Causes transmission to act as if an XOFF character has been received.
SETXON	Causes transmission to act as if an XON character has been received.

Returns

This return value is zero if successful. Otherwise, it is negative if the *fcn* parameter does not specify a valid function code.

Comments

The export ordinal for this function is 9.

If the communications driver does not export the **CommWriteString** function, Windows 3.1 intercepts and adjusts the return values for the GETBASEIRQ, GETMAXBAUD, GETMAXLPT, and GETMAXCOM functions after the **cextfcn** function returns.

LONG cextfcn(*cid*, *fcn*);

int *cid;*
int *fcn;*

The **cextfcn** function carries out the extended communications function specified by the *fcn* parameter. USER calls this function when an application calls the **EscapeCommFunction** function (USER.214).

Parameter	Description
cid	Identifies the communications device.
fcn	Specifies the extended function to carry out. It can be one of the following values.

Value	Meaning
CLRDTR	Clears the data-terminal-ready (DTR) signal.
CLRRTS	Clears the request-to-send (RTS) signal.
GETBASEIRQ	Returns the base-port address and IRQ setting for the COM port specified by the *cid* parameter. The low 16 bits of the return address specifies the base-port address, and the high address specifies the IRQ selection. If the high 16 bits is -1, then the port does not exist; if it is zero, the installed COMM driver does not support this escape.
GETMAXBAUD	Returns a constant that indicates the highest baud rate supported by the port specified by the *cid* parameter. The following constants may be returned.

Value	Meaning
CBR_110	The highest baud rate is 110.
CBR_300	The highest baud rate is 300.
CBR_600	The highest baud rate is 600.
CBR_1200	The highest baud rate is 1200.
CBR_2400	The highest baud rate is 2400.
CBR_4800	The highest baud rate is 4800.
CBR_9600	The highest baud rate is 9600.
CBR_19200	The highest baud rate is 19,200.
CBR_38400	The highest baud rate is 38,400.
CBR_56000	The highest baud rate is 56,000.
CBR_128000	The highest baud rate is 128,000.
CBR_256000	The highest baud rate is 256,000.

GETMAXCOM	Returns the maximum COM port identifier supported by the communications driver. This value ranges from 0x00 to 0x7F, such that 0x00 corresponds to COM1, 0x01 to COM2, 0x02 to COM3, and so on. The communications driver included in Windows 3.1 supports COM1 through COM4, and will always return 0x03.
GETMAXLPT	Returns the maximum LPT port ID supported by the system. This value ranges from 0x80H to 0xFFH, such that 0x80H corresponds to LPT1, 0x81H to LPT2, 0x82H to LPT3, and so on.
RESETDEV	Resets the printer device (that is, assert the reset line) if the *cid* parameter specifies an LPT port. No function is performed if *cid* specifies a COM port.
SETDTR	Sets the data-terminal-ready (DTR) control line on.
SETRTS	Sets the request-to-send (RTS) control line on.

SETXOFF	Causes transmission to act as if an XOFF character has been received.
SETXON	Causes transmission to act as if an XON character has been received.

Returns

This return value is zero if successful. Otherwise, it is negative if the *fcn* parameter does not specify a valid function code.

Comments

The export ordinal for this function is 9.

If the communications driver does not export the **CommWriteString** function, Windows 3.1 intercepts and adjusts the return values for the GETBASEIRQ, GETMAXBAUD, GETMAXLPT, and GETMAXCOM functions after the **cextfcn** function returns.

WORD cflush(*cid*, *q*)

int *cid;*
int *q;*

The **cflush** function flushes all characters from the transmit or receive queue of the specified communications device.

Parameter	Description
cid	Identifies the communications device.
q	Specifies which queue to flush. If the *q* parameter is 1, the function flushes the receive queue; if *q* is zero, it flushes the transmit queue.

Returns

The return value is the most recent error value.

Comments

The export ordinal for this function is 10.

int CommWriteString(*cid, pbuf, size*)

int *cid;*
LPSTR *pbuf;*
int *size;*

The **CommWriteString** function transmits a string of characters using the specified communications device. USER calls this function whenever an application calls the **WriteComm** function (USER.205).

Parameter	**Description**
cid	Identifies the communication device.
pbuf	Points to the buffer containing the bytes to write.
size	Specifies the number of bytes to write.

Returns

The return value is the number of bytes actually written.

Comments

The export ordinal for this function is 19.

During initialization, USER checks for this function to determine whether the driver supports the extended functions new to Windows 3.1. If the function is not present, Windows assumes that the driver is a Windows 3.0 driver and makes sure that all interaction with the driver is compatible with Windows 3.0.

int csetbrk(*cid*)

int *cid;*

The **csetbrk** function suspends character transmission and places the communications device in a break state. USER calls this function whenever an application calls the **SetCommBreak** function (USER.210).

Parameter	Description
cid	Identifies the communications device.

Returns

The return value is zero if the function is successful. Otherwise, the return value is -1.

Comments

The export ordinal for this function is 13.

See Also

cclrbrk

WORD ctx(*cid*, *ch*)

int *cid;*
int *ch;*

The **ctx** function places a character in a special holding variable to ensure it will be the next character to be transmitted. The function will return an error code if the special holding variable already contains a character. USER calls this function whenever an application calls the **TransmitCommChar** function (USER.206).

Parameter	Description
cid	Identifies the communications device.
ch	Specifies the character to transmit.

Returns

The return value is zero if the function is successful. The return value is one of the following values if there is an error.

Value	Meaning
0x8000	The function received an invalid *cid* parameter.
0x4000	The function could not write to the parallel port or there's already a character waiting for immediate transmission.

Comments

The export ordinal for this function is 6.

See Also

sndcom

int EnableNotification(*cid, hWnd, wInTrigger, wOutTrigger***)**

int *cid;*
HWND *hWnd;*
WORD *wInTrigger;*
WORD *wOutTrigger;*

The **EnableNotification** function enables or disables communications message posting. When enabled, the driver posts the **WM_COMMNOTIFY** message to the specified window. USER calls this function when an application calls the **EnableCommNotification** function (USER.245).

Parameter	Description
cid	Identifies the communication device.
hWnd	Identifies the window to receive the **WM_COMMNOTIFY** message. If this parameter is NULL, the function disables the notification.
wInTrigger	Specifies the minimum number of bytes to be received in the communication device's input buffer before receive notification is sent.
wOutTrigger	Specifies the maximum number of bytes to remain in the communication device's output buffer before transmit notification is sent.

Returns

The return value is TRUE if successful. Otherwise, the return value is FALSE.

Comments

The export ordinal for this function is 100.

The **WM_COMMNOTIFY** message has the following parameters.

Parameter	Description
wParam	Specifies the communication-device identifier (the *cid* parameter).
HIWORD(*lparam*)	Not used; must be zero.
LOWORD(*lparam*)	Specifies the notification status. It can be one of the following values.

Value	Meaning
CN_EVENT	An event enabled in the communication device's event mask (specified by the **SetCommEventMask** function) has occurred. The application should call the function **GetCommEventMask** to determine what event has occurred, and to clear the event. This status is sent when the communication device's event word changes. The application clears the appropriate event to ensure notification on subsequent events.
CN_RECEIVE	At least *wInTrigger* bytes are in the communication device's input buffer, or at least 1 byte is in the input buffer. Additionally, no more have been received before the end of an internal timeout period. The number of bytes in the input buffer must be lower than *wInTrigger* bytes before this message will be sent again.
CN_TRANSMIT	Fewer than *wOutTrigger* bytes remain in the communication device's output buffer to be transmitted. The number of bytes in the output buffer must exceed *wOutTrigger* bytes before this message will be sent again.

The communication device event may be a line-status or printer error. Applications can determine the cause by using the **GetCommError** function immediately after the **GetCommEventMask** function

(USER.209).

LPDCB getdcb(*cid*)

int *cid;*

The **getdcb** function retrieves the device-control block (DCB) for the specified device. USER calls this function whenever an application calls the **GetCommState** function (USER.202).

Parameter	Description
cid	Identifies the communications device.

Returns

The return value is a pointer to a DCB structure.

Comments

The export ordinal for this function is 15.

See Also

DCB

WORD inicom(*lpdcb*);

DCB FAR **lpdcb;*

The **inicom** function initializes the specified communications device. USER calls this function whenever an application calls the **OpenComm** function (USER.200).

Parameter	**Description**
lpdcb	Points to a DCB structure.

Returns

The return value is zero if the function is successful. Otherwise, the return value is a nonzero (IE_*) value if there is an error.

Comments

The export ordinal for this function is 1.

See Also

trmcom, DCB

void ReactivateOpenCommPorts(*void*)

The **ReactivateOpenCommPorts** function reactivates open communication ports. WINOLDAP uses this function to reactivate communications ports after switching back to Windows.

Returns

This function has no return value.

Comments

The export ordinal for this function is 18.

See Also

<u>SuspendOpenCommPorts</u>

int reccom(*cid*)

int *cid;*

The **reccom** function reads a byte from a communications device. USER calls this function whenever an application calls the **ReadComm** function (USER.204).

Parameter	**Description**
cid	Identifies the communications device.

Returns

The return value is the byte read. Otherwise, the return value is -2 if no data is available, and -1 if there is an error.

Comments

The export ordinal for this function is 4.

See Also

sndcom

WORD setcom(*lpdcb*);

DCB FAR **lpdcb;*

The **setcom** function sets a communication device to the state specified by the DCB structure. USER calls this function whenever an application calls the **SetCommState** function (USER.201).

Parameter	Description
lpdcb	Points to a DCB structure.

Returns

The return value is zero if the function is successful. Otherwise, it is a negative (IE_*) value if an error occurs.

Comments

The export ordinal for this function is 2.

This function reinitializes all hardware and controls as defined by the *lpdcb* parameter, but does not clear transmit or receive queues.

See Also

DCB

int setque(*cid*, *lpqdb*)

int *cid;*
qdb FAR * *lpqdb;*

The **setque** function sets the transmit and receive queues. USER calls this function whenever an application calls the **OpenComm** function (USER.200).

Parameter	Description
cid	Identifies the communications device.
lpqdb	Points to a **qdb** structure.

Returns

The return value is zero if the function is successful. Otherwise, the return value is IE_BADID if the *cid* parameter is not valid.

Comments

The export ordinal for this function is 3.

See Also

qdb

WORD sndcom(*cid*, *ch***)**

int *cid;*
int *ch;*

The **sndcom** function writes a byte to the specified communications device. USER calls this function whenever an application calls the **WriteComm** function (USER.205).

Parameter	Description
cid	Identifies the communications device.
ch	Specifies the character to transmit.

Returns

The return value is zero if the function is successful. Otherwise, the return value is a nonzero value if there is an error.

Comments

The export ordinal for this function is 5.

See Also

reccom

WORD stacom(*cid*, *cs*)

int *cid;*
COMSTAT FAR **cs;*

The **stacom** function returns the most recent error value and copies the current status for the specified device to the given structure. USER calls this function whenever an application calls the **GetCommError** function (USER.203).

Parameter	Description
cid	Identifies the communications device.
cs	Points to a COMSTAT structure.

Returns

The return value is the most recent error code.

Comments

The export ordinal for this function is 8.

See Also

COMSTAT

void SuspendOpenCommPorts(*void*)

The **SuspendOpenCommPorts** function suspends open communication ports. WINOLDAP uses this function to suspend communications ports before switching to non-Windows applications.

Returns

This function has no return value.

Comments

The export ordinal for this function is 17.

See Also

ReactivateOpenCommPorts

int trmcom(*cid*)

int *cid;*

The **trmcom** function closes the specified communications device and frees any memory allocated for the device's transmit and receive queues. All characters in the output queue are transmitted before the communications device is closed unless an error occurs. The **trmcom** function returns an error immediately if the *cid* parameter is not valid, and returns an error if a timeout occurs while attempting to send the characters.

USER calls this function whenever an application calls the **CloseComm** function (USER.207).

Parameter	Description
cid	Identifies the communications device.

Returns

The return value is zero if the function is successful. The function returns -2 if an error occurred while transmitting the remaining characters from the transmit queue. If the *cid* parameter is not valid, **trmcom** will return 0x8000.

Comments

The export ordinal for this function is 7.

If the return value is -2, the function may have left some characters in the transmit queue, but may have successfully sent others. If an application subsequently calls the **stacom** through the **GetCommError** function (USER.203), the driver should set the **cbOutQue** member in the COMSTAT structure to allow the application to determine how many characters were actually transmitted.

See Also

inicom

void SuspendOpenCommPorts(*void*)

The **SuspendOpenCommPorts** function suspends open communication ports. WINOLDAP uses this function to suspend communications ports before switching to non-Windows applications.

Returns

This function has no return value.

Comments

The export ordinal for this function is 17.

See Also

ReactivateOpenCommPorts

int trmcom(*cid*)

int *cid;*

The **trmcom** function closes the specified communications device and frees any memory allocated for the device's transmit and receive queues. All characters in the output queue are transmitted before the communications device is closed unless an error occurs. The **trmcom** function returns an error immediately if the *cid* parameter is not valid, and returns an error if a timeout occurs while attempting to send the characters.

USER calls this function whenever an application calls the **CloseComm** function (USER.207).

Parameter	Description
cid	Identifies the communications device.

Returns

The return value is zero if the function is successful. The function returns -2 if an error occurred while transmitting the remaining characters from the transmit queue. If the *cid* parameter is not valid, **trmcom** will return 0x8000.

Comments

The export ordinal for this function is 7.

If the return value is -2, the function may have left some characters in the transmit queue, but may have successfully sent others. If an application subsequently calls the **stacom** through the **GetCommError** function (USER.203), the driver should set the **cbOutQue** member in the COMSTAT structure to allow the application to determine how many characters were actually transmitted.

See Also

inicom

```
typedef struct tagCOMSTAT {
  BYTE fCtsHold: 1;
  BYTE fDsrHold: 1;
  BYTE fRlsdHold: 1;
  BYTE fXoffHold: 1;
  BYTE fXoffSent: 1;
  BYTE fEof: 1;
  BYTE fTxim: 1;
  WORD cbInQue;
  WORD cbOutQue;
} COMSTAT;
```

The **COMSTAT** structure contains information about a communications device.

Member	Description
fCtsHold	Specifies whether transmission is waiting for the clear-to-send (CTS) signal to be sent.
fDsrHold	Specifies whether transmission is waiting for the data-set-ready (DSR) signal to be sent.
fRlsdHold	Specifies whether transmission is waiting for the receive-line-signal-detect (RLSD) signal to be sent.
fXoffHold	Specifies whether transmission is waiting as a result of the XOFF character being received.
fXoffSent	Specifies whether transmission is waiting as a result of the XOFF character being transmitted. Transmission halts when the XOFF character is transmitted and used by systems that take the next character as XON, regardless of the actual character.
fEof	Specifies whether the end-of-file (EOF) character has been received.
fTxim	Specifies whether a character previously passed to the **ctx** function is waiting to be transmitted.
cbInQue	Specifies the number of characters in the receive queue.
cbOutQue	Specifies the number of characters in the transmit queue.

See Also

stacom

```
typedef struct {
   char  Id;            /* internal device ID              */
   ushort Baudrate;      /* operating speed                */
   char  ByteSize;      /* transmit/receive byte size      */
   char  Parity;       /* 0,1,2,3, or 4                   */
   char  StopBits;      /* number of stop bits             */
   ushort RlsTimeout;    /* timeout for RLSD to be set      */
   ushort CtsTimeout;    /* timeout for CTS to be set       */
   ushort DsrTimeout;    /* timeout for DSR to be set       */
   ushort fBinary: 1;    /* binary-mode flag                */
   ushort fRtsDisable: 1; /* disable RTS                    */
   ushort fParity: 1;    /* enable parity checking          */
   ushort fDummy: 5;
   ushort fOutX: 1;      /* enable output XON/XOFF          */
   ushort fInX: 1;       /* enable input XON/XOFF           */
   ushort fPeChar: 1;    /* enable parity-error replacement */
   ushort fNull: 1;      /* enable null stripping           */
   ushort fChEvt: 1;     /* enable Rx character event       */
   ushort fDtrflow: 1;   /* enable DTR flow control         */
   ushort fRtsflow: 1;   /* enable RTS flow control         */
   ushort fDummy2: 1;
   char  XonChar;       /* transmit/receive XON character  */
   char  XoffChar;      /* transmit/receive XOFF character */
   ushort XonLim;        /* transmit XON threshold          */
   ushort XoffLim;       /* transmit XOFF threshold         */
   char  PeChar;        /* parity error replacement character */
   char  EofChar;       /* end-of-input character          */
   char  EvtChar;       /* event-generating character      */
   ushort TxDelay;       /* amount of time between characters */
} DCB;
```

The **DCB** structure contains the RS-232 configuration parameters for a communication device.

Member	Description
Id	Specifies the device ID byte (COM1 = 0, COM2 = 1, and so on). This is also the value returned by the **cOpen** function, when successful.
Baudrate	Specifies the operating speed; any baud rate supported by the hardware.
ByteSize	Specifies the transmitting and receiving byte size; normally in the range 4 through 8.
Parity	Specifies the parity setting. The value can be one of the following values.

Value	Meaning
0	None
1	Odd
2	Even
3	Mark
4	Space

Member	Description
StopBits	Specifies the number of stop bits. The value can be one of the following values.

Value	Meaning
0	1 stop bit
1	1.5 stop bits
2	2 stop bits

Member	Description
RlsTimeout	Specifies the amount of time, in milliseconds, to wait for receiving-line-signal detect

	(RLSD) to become high. RLSD flow control can be achieved by specifying infinite timeout (0xFFFF).
CtsTimeout	Specifies the amount of time, in milliseconds, to wait for clear-to-send signal (CTS) to become high. CTS flow control can be achieved by specifying infinite timeout (0xFFFF).
DsrTimeout	Specifies the amount of time, in milliseconds, to wait for data-set-ready (DSR) to become high. DSR flow control can be achieved by specifying infinite timeout (0xFFFF).
fBinary	Specifies the binary-mode flag (0 is ASCII mode, 1 is binary). In ASCII mode, the end-of-file character (EOFCHAR) is recognized and remembered as the end of received data.
fRtsDisable	Disables the receive-transmission signal (RTS) line for as long as this device is open, if set. Normally, RTS is enabled when the device is opened and disabled when closed.
fParity	Enables parity checking, if set.
fOutX	Indicates that XON/XOFF flow control is to be used during transmission, if set. The transmitter halts when it receives an XOFF character, and starts again when it receives an XON character.
fInX	Indicates that XON/XOFF flow control is to be used during reception, if set.
fPeChar	Indicates that characters received with parity errors are to be replaced with the specified parity-checking characters (PECHAR), if set.
fNull	Specifies that the received null characters are to be discarded, if set.
fChEvt	Indicates that the reception of event-checking characters (EVTCHAR) are to be flagged as an event, if set.
fDtrFlow	Indicates that the data-terminal-ready signal (DTR) is to be used for receive flow control, if set.
fRtsflow	Indicates that the receive-transmission signal (RTS) is to be used for receive flow control, if set.
XonChar	Specifies the XON character for both transmit and receive.
XoffChar	Specifies the XOFF character for both transmit and receive.
XonLim	Specifies the threshold value for receive queue. When the receive queue comes within 10 characters of being full, it transmits an XOFF character. When the queue comes within 10 characters of being empty, an XON character will be transmitted.
XoffLim	Specifies the threshold value for send queue. When the number of characters in the receive queue exceeds this value, an XOFF character is sent (if XOFF flow control is enabled) and the data-terminal-ready signal (DTR) is dropped (if enabled).
PeChar	Specifies the character to be used as replacement when a parity error occurs.
EofChar	Specifies the character that signals the end of the input.
EvtChar	Specifies the character that triggers an event flag.
TxDelay	Specifies the minimum amount of time that must pass between transmission of characters.

See Also

getdcb, inicom, setcom

```
typedef struct {
   char far    *pqRx;  /* pointer to Rx queue          */
   int         cbqRx;  /* size of Rx queue in bytes    */
   char far    *pqTx;  /* pointer to Tx queue          */
   int         cbqTx;  /* size of Tx queue in bytes    */
} qdb;
```

The **qdb** structure contains information about the location and size of the transmit and receive queues.

Member	Description
pqRx	Points to the receive queue.
cbqRx	Specifies the size (in bytes) of the receive queue.
pqTx	Points to the transmit queue.
cbqTx	Specifies the size (in bytes) of the transmit queue.

See Also

setque